CONF-870310--1

LA-UR -87-1038

TITLE: IMPROVING PRODUCTIVITY WITH A CCC TOOL KIT

LA-UR--87-1038

DE87 007507

AUTHOR(S): Gary Carr, LANSCE

SUBMITTED TO: To be presented and published in proceedings of the Spring Meeting of Softool Users Group March 30-31, 1987 Santa Barbara, California

IMPROVING PRODUCTIVITY WITH A CCC TOOL KIT

Gary Cort
Los Alamos National Laboratory
Los Alamos, New Mexico 87544

Introduction

In the years since its introduction, the CCC configuration management environment has established itself as the premiere CM product for projects of all sizes. The functionality and flexibility offered by this system can be employed to provide automated support for essentially every facet of a program of software (or hardware) configuration management, and, as a bonus, can streamline may software development activities. The large number of CCC-based CM/support/development environments[1-10] which have already been developed for numerous diverse applications bear silent witness to the effectiveness and adaptability of the CCC product.

It must be emphasized, however, that the bare CCC product does not constitute a standalone, automated, turn-key configuration management system, but simply provides a secure environment and a comprehensive set of low-level tools from which such a system can be built. Indeed, the CCC macro language provides high-level structured constructs as well as access to all of the advanced facilities of the CCC environment, thereby to support construction of automated CM systems which are highly tailored to the implementing organization's requirements. Consequently, most of the CCC applications which are reported in the literature are highly customized systems which are specially designed and implemented to accomodate an organization's development and CM environment.

Although customized CCC applications are of general interest from the perspective of general functionality as well as philosophy and strategy of CCC utilization, they are also highly specialized to address the specific environment of the organization for which they are developed. Consequently, systems such as these are of little practical benefit to other organizations for which the operational environment may be considerably different. In addition, large customized applications are often written in a manner which makes them difficult to modify, thereby presenting significant maintenance difficulties as the application environment evolves.

This paper presents a strategy for developing CCC applications based around a kernel of small, modular, and cohesive utility macros. Such utilities can be employed as building blocks from which a larger, customized application can be constructed. This approach emphasizes the identification of low-level activities and operations which are fundamental to the application and packages them as reusable software components. This reliance on reusability promotes designs which are conceptually simpler, introduces a level of uniformity and consistency which facilitates maintenance activities, and can make significant contributions to reducing the overall size of the final CCC application. In addition, it is often the case that an application which is implemented as a hierarchy of small modules can make much more effective use of CCC features (*e.g.* index data structures) than can a system which is monolithically implemented the utility macros themselves will transfer easily between environments and projects, thereby providing a solid foundation around which subsequent applications may be built.

Hints and Guidelines for Developing Utility Macros

For a utility macro to be effective within a larger application, it must exhibit two general qualities: reusability and uniformity. Consequently, the hints and guidelines identified in the following paragraphs are chosen to enhance these features. Reusability, of course, is the ultimate goal of the utility macro designer. The degree of reusability of of a utility macro is determined primarily by modularity and cohesiveness issues although uniform and comprehensive users documentation can also contribute significantly. Uniformity issues generally affect the overall maintainability of a suite of utility macros and are therefore crucial to the overall robustness of the system.

Design for reusability. Probably the most important aspect of designing a CCC application is to design for reusability. This strategy encourages the designer(s) to identify potential utility applications at the earliest stages of the application design, and to make a conscientious effort to exploit their reusability at every suitable opportunity. This approach guarantees that your utilities will constitute a consistent set of highly functional units rather than a collection of haphazardly implemented contrivances.

Ensure cohesiveness. The most effective utility macros are those that perform a single, well defined, function, rather than a collection of logically unrelated or disjoint actions. Because of their simplicity, cohesive macros are conceptually easier to design, as well as significantly easier to test and debug. Generally, a macro which exhibits a high level of cohesion will be useful in a wider variety of situations than will a less cohesive counterpart. Additionally, a cohesive macro is much less likely to be affected by changes in the application environment than is a more complex, less cohesive implementation.

Eliminate global references. Global references complicate macro interfaces and generally reduce the robustness of your applications. Never use global references to communicate between macros, use macro parameters instead. This approach improves the understandability and reliability of your macros and fosters a level of generality that makes your utilities far easier to reuse. Also, restrict the scope of internal variables and temporary texts so that they are not visible outside the utility macro. In so doing, you eliminate the possibility of problems with unexpected side effects.

Provide adequate documentation. Regardless of the cohesiveness and modularity attributes of a utility macro, if it is impossible to determine what it is supposed to do and/or how to interface to it, no one will use it. In addition, if maintenance-level documentation is missing or inadequate, it is unlikely that the macro will be updated as requirements or the operational environment change. Documentation is particularly important in the CCC environment because of the extreme complexity of the CCC macro language. This high level of complexity can make even a short macro very difficult to understand from the source code alone. It is therefore recommended that two levels of inline documentation be provided for every macro: uniform, comprehensive users' documentation and detailed annotation of the source code. Users documentation is most effective which is present at the beginning of the macro. It should provide a brief statement of the purpose of the macro, a description of the parameters (inputs and outputs), a list of assumptions and limitations which constrain the applicability or operational environment of the macro and an example of how the macro is invoked. A standard template may be employed to guarantee the uniformity and completeness of this documentation. Maintenance documentation should be interspersed throughout the source code and should describe local variables as well as the various operations being performed. Maintenance documentation should be included at least every three to five lines--more often if extremely complex operations such as multiple level symbol substitutions are being performed.

Use a consistent style. The general level of readability of your macros contributes significantly to their ultimate success or failure. Application of simple style rules will decrease the level of effort required to read and understand these modules, thereby making them easier to reuse and maintain. Define a simple set of standards for coding and documentation style. Employ meaningful names throughout and be consistent between macros. In particular, use symbolic names for parameters rather than the less meaningful &1, &2, *etc*. Establish rules for using upper and lower case within your macros. Decide on an indentation scheme for structured constructs and stick with it. Partition your macros into logical units which segregate variable declarations, initializations, and cleanup operations from the functional portions of the macro. Ensuring uniformity through the use of these and other simple style conventions is an investment in the longevity, robustness and friendliness of your utility software.

A Typical Utility Macro Tool Set

Many criteria exist for determining the contents of your macro tool kit. Experience indicates, however, that satisfaction of either of the following two conditions is generally sufficient to bestow utility status upon a macro: (a) the macro has a wide audience of users any of whom will make occasional use of it, and (b) the macro has a restricted audience but is invoked frequently therein. The former case usually corresponds to a macro which extends the overall functionality of the CCC environment and therefore accrues a large number of potential users. The second class of macro often constitutes a set of automated procedures employed by a particular class of users (*e.g.* data base administrators or software managers) to perform repeatedly a particular maintenance function.

The following paragraphs describe the functionality of selected utility macros which have been implemented by the author. These macros have been chosen to demonstrate the wide variety of applications which can be addressed by a suite of small, independent utilities. Included among these examples are utilities of each general class.

Within the class of macros which extend the overall functionality of the CCC environment are the following utilities: **SIMPORT** and **SEXPORT** to provide powerful, syntax-directed import and export functions, **EDIT_HOST** a simple, yet flexible, utility which permits CCC-resident texts to be edited with a host editor, **INITIATE_LOGIN** which automatically executes a hierarchy of user- and manager-defined login texts at the time at which a user logs into CCC, and **BATCH**, a utility which permits CCC macros to be executed in batch mode from within CCC. Within the class of macros which automate maintenance activities are **AUTHORIZE_USER**, a utility which automates the process of user authorization as well as traits and access control definition, and **ENABLE_DSM**, a macro which supports the structured maintenance of the data structure macros which comprise a CCC application.

One of the most common operations performed during any CCC session involves the import or export of modules between the host file system and the CCC data base. Often it is necessary to move large collections of modules with a single wildcarded IMPORT or EXPORT command. Unfortunately, these CCC commands do not support a very sophisticated algorithm for interpreting wildcarded specifications so, except in the simplest of cases, the operation may fail with a run-time error. Consider, for example, an export operation which seeks to move all CCC texts which satisfy the wildcard specification *.PAS from the current data structure to the VMS directory [.DESTINATION]. In order to perform this operation the following CCC command might be used

EXPORT FROM=*/PAS TO=[.DESTINATION]*.* OPTION=READ

Shortly after executing this command, however, the user is notified by the CCC run time system that the destination specification cannot be interpreted and the command aborts. Indeed, the only way to succeed in this endeavor is to issue the command:

EXPORT FROM=*/PAS TO=*.* OPTION=READ

and to ensure that [.DESTINATION] is the default VMS directory (*i e* the VMS directory from which you logged into CCC) The IMPORT command exhibits similar problems in interpreting complex wildcard specifications. Needless to say, this feature can make moving large numbers of modules between the host file system and the CCC data base an extremely tedious operation which requires multiple login/logout sequences

The syntax-directed import and export utility macros **SIMPORT/DSM** and **SEXPORT/DSM** contain the necessary intelligence respectively to export CCC texts to a nondefault VMS directory, or to import files from a nondefault directory. Using **SEXPORT** the above operation is performed as follows:

SEXPORT */PAS [.DESTINATION] READ

SEXPORT causes all CCC texts which match the specification provided in the first parameter to be exported to the VMS directory specified in the second parameter. The destination directory specification may optionally include a device specification (*e g* DUA0:[CORT.DSMS]): the utility macro automatically validates the existence of the destination and checks for user access thereto. Note that **SEXPORT** forces the names and extensions of the exported files to be identical to their CCC-resident counterparts. Additionally, the **SEXPORT** interface is specified in a manner which is consistent with the EXPORT command, thereby enhancing uniformity and simplifying human interactions. **SEXPORT** therefore provides the functionality necessary to extend the existing CCC EXPORT command to process the more general case for which a nondefault VMS directory specification is provided as the export destination.

Similar functionality is provided for the **SIMPORT** utility. A typical invocation of this macro is shown below.

SIMPORT USER_DISK:[.DESTINATION]* FOR CCCSYSTEM CONFIG NEW NO

where the first parameter specifies the VMS files to be imported, the second parameter indicates the CCC data structure which is the destination, the third parameter is the import option and the last parameter bypasses prompting Similar assumptions to above are made concerning file names and extensions Once again, the interface is defined to be consistent with the CCC IMPORT command As such, the **SIMPORT** and **SEXPORT** utilities provide a logically consistent set of commands which extend the capabilities of the corresponding native CCC commands

EDIT_HOST/DSM is a utility macro application which permits a CCC user to edit a CCC-resident text with a host operating system editor without the necessity of performing explicit import and export operations **EDIT_HOST** accepts the specification of the CCC text to be edited as its only parameter, automatically exports the text to the host operating system and starts an interactive HOST editing session using a preselected host editor. In the event that the text to be edited does not yet exist within CCC, it is automatically created At the conclusion of the editing session, **EDIT_HOST** automatically reimports the edited text, updates the appropriate change auditing information, and deletes the host-resident version of the edited text.

This utility is an example of an extremely simple application which provides enormous functionality to the user community. It allows ordinary users to employ powerful and familiar host resident screen-oriented editors in their daily work rather than being forced to use the CCC line editor. The interface effectively hides the intermediate import, export and file deletion operations, thereby simplifying human interactions. A simple enhancement will permit a user to specify the editor of choice rather than use the preselected editor specified by **EDIT_HOST**.

The **BATCH/DSM** utility macro is an example of a more complex application which significantly increases the flexibility of the CCC environment. This utility permits a user to initiate the batch execution of a CCC macro from within a CCC session. Recall that the unenhanced CCC environment supports two methods for accessing host operating system batch queues. The first mode utilizes the CCCBATCH facility to execute CCC macros in batch mode. However, this facility can only be invoked from the host operating system: it is inaccessible from within a CCC session. The second mode employs the HOST facility to initiate a batch job from within a CCC session. In this case, however, the batch job must consist of a host command file (*e.g.*, a collection of DCL commands) rather than a CCC macro. As such, the means for executing a CCC macro in batch mode from a CCC session does not exist.

The **BATCH** utility was implemented to address this deficiency. It accepts as a parameter the name of the CCC text to be executed in batch mode and uses the CCC HOST AUTOMATIC command in conjunction with CCCBATCH to submit the job directly from the CCC session. Other parameters provided to the utility specify the data base in which the batch macro shall execute, as well as the host batch queue to which the job is submitted. Standard defaults are specified within the utility so that if the data base parameter is omitted, the job executes within the data base from which it is submitted. Another default is defined to handle the case in which the batch queue is omitted from the parameter list.

This utility provides a range of options and level of convenience to the ordinary user that can significantly streamline a CCC session. Long, noncritical jobs can easily be processed in background mode without undue interruptions to the interactive session. It also makes feasible an entire range of more esoteric applications, *e.g.*, the **BATCH** utility provides a viable mechanism for communicating between data bases.

The last of the globally useful utilities to be discussed in this monograph is a simple system for implementing a VMS-like login facility. One of the very useful features of the VMS operating system is the capability to execute automatically a hierarchy of "login" files at the time that a user logs into the system. This facility generally executes a global login file (provided by system management personnel) first, followed by the execution of a locally-defined file. The login files are used to define entities which are generally useful throughout the user session. The global file is usually employed to make system-wide definitions, whereas the local file is employed further to customize the user environment.

The CCC environment can benefit significantly from a similar facility. Indeed, because a typical CCC user may be subordinate to many levels of managers, a facility which supports an extended hierarchy of login texts is probably appropriate. Unfortunately, the CCC system is configured to permit only one level of login text to be executed. The login text, in addition, must be located within the user's login data structure, thereby virtually eliminating the feasibility of providing manager definitions therein.

In order to remedy this situation, the author has implemented utility macros to define a local LOGIN facility which automatically executes a hierarchy (of virtually unlimited depth) of manager- and user-supplied login texts at user login time. Each text resides within the scope of the defining user or manager, and outside the scope of any subordinates. Consequently, each user or manager can configure their particular login text in a straightforward manner without encroaching on anyone else's domain. Each login text is given the standard name **LOGIN/DSM** and resides in the login data structure of the defining user or manager.

The utility macro **START_LOGIN_SEQUENCE/DSM** is physically copied into the login data structure of each user who is authorized to access the CCC data base. The SETENVIRONMENT command is employed to set the login macro for each user to this DSM As such, upon initiating a CCC login sequence, the user's local **START_LOGIN_SEQUENCE** is automatically executed. This recursive data structure macro, which is identical for each user, automatically searches for all **LOGIN/DSM** macros which are present in the user's login data structure as well as in any superordinate data structures. It then automatically executes the various **LOGIN/DSM**'s in order of decreasing scope. In this manner, any **LOGIN/DSM** macro which is resident at the DBA level is executed first, followed in order by **LOGIN/DSM**'s at the SYSTEM, CONFIGURATION and successive MODULE levels ending with execution of the **LOGIN/DSM** which resides within the user's login data structure.

The definitions made during the login sequence generally remain in force for the duration of the CCC session The potential uses for this facility are extensive and include providing short names for local macros, simplifying data base navigation, providing local defaults for symbols or macro parameters, simplifying access controls and producing reports at login time. Use of utility macros to implement this facility provides a simple, common interface, uniform functionality and ease of maintenance In addition it permits the various login texts to be physically isolated and distributed throughout the data base hierarchy in a manner which reflects their scope of effect

Among the class of system maintenance utility macros, two are discussed below. The utility macro **DEFINE_USER/DSM** was developed to automate and to consolidate the operations which must be performed in authorizing or removing a CCC user. The macro provides a prompted interface which requests information such as the name and class of the user, whether the user is to be added or deleted, the user's login data structure, and default access controls and data structure traits associated therewith. The utility then executes the appropriate CCC commands to perform the authorization operations as well as to configure the user's environment (*e.g.* place a default **LOGIN/DSM** and a copy of **START_LOGIN_SEQUENCE/DSM** in the login data structure) As such, this utility constitutes a high-level facility which dramatically improves the efficiency and reliability of performing user authorization operations.

ENABLE_DSM/DSM is the last utility macro to be discussed in the context of management aids This utility provides a facility for controlling the morphology and accessibility of data structure macros which of necessity must have a high visibility In general, most CCC applications are implemented as collections of data structure macros which reside at a high level (often at the data base level) of the CCC data base As such, these macros are visible (for execution) to all subordinate users Although this high visibility is desireable from the aspect of execution accessibility, it often causes problems when one or more of the macros is under development or modification.

For example, prudent development practices dictate that your data structure macros have the ARCHIVE=YES trait thereby to guarantee that a change history be maintained during development However, initiation of a macro with multiple archived revisions can be tediously slow, thereby resulting in a significant degradation of execution performance. Furthermore, even after the macro has stabilized and is no longer undergoing extensive changes, there may exist excellent reasons for retaining the archived versions.

Other problems may result from providing extensive inline documentation (as recommended in the preceding section) for macro applications. The CCC macro language executes in an interpreted mode, so extensive amounts of nonexecutable inline documentation may also degrade performance.

Finally, it is often necessary keep an old version in service while an existing macro is modified, or to hide a newly developed macro from the user community until such time as verification and validation is completed. These requirements argue for a separate repository for uncertified macros.

ENABLE_DSM supports the notion of separate repositories for uncertified and production-level macros. It presupposes the existence of a CCC data structure in which macros are developed and maintained and a separate partition from which they can be executed. The development/maintenance partition is defined in a manner such that it is not visible to any subordinate user. As such there is no danger that any macro which resides therein can be executed by the user community. This partition is configured with the appropriate traits (*e.g.* archiving is activated, optionally compression and/or encryption are enabled) to support development and maintenance activities.

The execution partition is located such that it is visible by the appropriate segments of the user community. In addition, it is configured to support rapid execution (e.g. no archiving, compression or encryption). The **ENABLE_DSM** utility is employed to promote macros from the development partition to the execution partition and to perform simple optimizations to improve execution performance. The first priority of the utility is to copy the specified macro(s) from the development partition into the execution partition. This provides a version of the macro in the execution partition which has the appropriate traits for fast initiation and execution. The second act of the utility is to delete inline documentation from the macro which is resident in the execution partition, thereby to further improve its execution performance.

The result of these operations is to provide a version of the macro in the execution partition which is configured for rapid execution yet to retain the fully documented, multi-revision version in the development partition. Furthermore, subsequent modifications can be made within the development partition without disturbing the executing version. Upon completion of the revisions, the new version can be promoted to the execution partition with a minimum of disruption to the operational environment. In this manner, the project remains virtually undisturbed and enhancements/bug fixes can be integrated in a simple, structured manner.

Conclusions

In the preceding sections we have attempted to demonstrate the wide range of functionality, flexibility and versatility which are available through the use of utility macros. By implementing these constructs as simple building blocks from which complex applications can be built, new levels of reliability and maintainability can be easily achieved at a relatively small incremental cost. Often, as in the case of the **BATCH** utility described above, the higher level of abstraction which is obtained through utility packaging results in innovative new applications. In any case, establishment of a suite of robust, cohesive tools with uniform and consistent interfaces will significantly increase overall productivity in implementing new applications and performing daily operations.

No attempt has been made to describe a comprehensive set of utility macros. Our purpose has been merely to provide a sampling of the wide variety of useful applications that can be implemented in this manner. Indeed, the collection of utilitties which should be implemented to support a particular environment depends strongly on many factors including the character of the application, experience and requirements of the user community, and the general level of familiarity of the developers with the CCC environment. Only one generalization can safely be made: creation of an effective CCC tool kit will improve the flexibility and effectiveness of your applications as well as the productivity of your project members.

References

1. "Document Management", D. Cheney, Proc. Softool Users' Group, (Santa Barbara, September, 1984).

2. "Configuration Management for Mission-Critical Software: The Los Alamos Solution", G. Cort and D. M. Barrus, Proc. Softool Users' Group, (Santa Barbara, September, 1984).

3. "The Los Alamos Automated Configuration Accounting System: A Proposal", G. Cort and J. A. Goldstone, Proc. Softool Users' Group, (Dallas, March, 1985).

4. "The Los Alamos Hybrid Environment: An Integrated Development/Configuration Management Environment", G. Cort in Conference on Software Tools, J. Manning, ed. (IEEE Computer Society Press, New York, 1985).

5. "A Development Methodology for Scientific Software", G. Cort, *et al* , IEEE Trans Nuclear Science, NS-32, 1439, (1985).

6. "The Los Alamos Tool Oriented Software Development System", G. Cort and R. O. Nelson,, Proceedings of the Fall U. S. DECUS Symposium, (DECUS, Anaheim, 1985).

7. "The Los Alamos Software Development Tools", G. Cort and R. O. Nelson,, Proceedings of the Fall U. S. DECUS Symposium, (DECUS, Anaheim, 1985).

8. "A Menu Method of Accessing CCC", D. Hahn, Proc. Softool Users' Group, (Culver City, April 1986).

9. "CCC/MVS ISPF Turnkey Application", L. Stanton, Proc. Softool Users' Group, (Culver City, April 1986).

10. "A Modular, Automated Software Testing Environment", G. Cort and J. A. Goldstone, Proc. Softool Users' Group, (Culver City, April 1986).